OwlExporter

Guide for Users and Developers

René Witte
Ninus Khamis

Release 3.2
September 28, 2014

Semantic Software Lab
Concordia University
Montréal, Canada
http://www.semanticsoftware.info
## Contents

1. **Introduction to the OwlExporter** 1  
   1.1. Ontology Population Foundations 1  
   1.2. OwlExporter Overview 2  
      1.2.1. OwlExporter Features 2  
   1.3. How to read this documentation 3  

2. **Your First Ontology Population Pipeline** 4  
   2.1. OwlExporter Installation 4  
      2.1.1. Loading the Demo Application 5  
      2.1.2. Running the Demo Ontology Population Pipeline 5  
      2.1.3. Result – The Populated Ontology 7  
   2.2. OwlExporter Concepts 7  
      2.2.1. Exporting Domain Individuals 7  
      2.2.2. Exporting Domain Datatype Relationships 9  
      2.2.3. Exporting Domain Object Property Relationships 9  
      2.2.4. Exporting NLP Individuals 10  
      2.2.5. Exporting NLP Datatype Relationships 12  
      2.2.6. Exporting NLP Object Property Relationships 13  
      2.2.7. Exporting Object Property Relationships across Domain and NLP Ontologies 13  

3. **OwlExporter Reference** 15  
   3.1. OwlExporter Run-time Parameters 15  
   3.2. Advanced Features 15  
      3.2.1. Document Annotation 15  
      3.2.2. Exporting Coreference Chains 16  

4. **OwlExporter Implementation Notes** 18  
   4.1. Compilation 18  
   4.2. Importing the OwlExporter 18  
   4.3. Developer Notes 19  
      4.3.1. Duplicate Instances 19  
      4.3.2. Annotation Types versus Annotation Sets 19  

**OwlExporter Appendix** 20  
A. Main Multi-Phased Jape 20  
B. Doc Info Jape 20  
C. Mention Map Domain Entities 20  
D. Mention Map NLP Entities 21  
E. Mention Map Domain Relation 22  
F. Mention Map NLP Relation 22  
G. Mention Map Domain NLP Relation 23
Contents

Bibliography 24
List of Figures

1.1. Populating Ontologies from Text .............................................. 1
1.2. The OwlExporter Processing Resource ..................................... 2

2.1. Enabling the Semantic Software Lab repository in GATE’s CREOLE Plugin Manager 4
2.2. Loading the OwlExporter pipelines in GATE Developer .................. 5
2.3. The ANNIE and NLP Ontologies .................................................. 6
2.4. The Demo Document in GATE Developer ..................................... 7
2.5. Populated Ontology ................................................................. 8
2.6. Populated NLP Ontology ........................................................... 11
2.7. Example Annotations for NLP Ontology Export .......................... 12

3.1. An Example of the Doc Info Grammar ........................................ 16

4.1. GATE Known CREOLE Window .................................................. 19

About this document

This is the documentation for the OwlExporter, a component for the GATE architecture to export the results of a text analysis pipeline into an OWL ontology. Please refer to http://www.semanticsoftware.info/owlexporter for more information.

Acknowledgments

The following developers contributed to the design and implementation of the OwlExporter: René Witte, Ninus Khamis, Elian Angius, Qiăngqiang Li and Thomas Kappler.

Licenses

The OwlExporter is published under the GNU Affero General Public License v3 (AGPL3).1 Different licensing apply to resources distributed with the OwlExporter example pipeline:

ANNIE and ANNIE Ontology (domain.owl): GNU LESSER GENERAL PUBLIC LICENSE (LGPL) 2

1AGPL3, http://www.gnu.org/licenses/agpl-3.0.html
2LGPL, http://www.gnu.org/licenses/lgpl.html
List of Figures

Protégé: MOZILLA PUBLIC LICENSE (MPL) ³

³MPL, http://www.mozilla.org/MPL/
Chapter 1.

Introduction to the OwlExporter

In this chapter, we provide a brief introduction to the concept of ontology population from text and explain the major features of our OwlExporter GATE PR. For more background information, please refer to our publication (Witte et al., 2010).

1.1. Ontology Population Foundations

Ontologies in OWL format (Baader et al., 2003) provide for a standardized means of modelling, querying, and reasoning over large knowledge bases. Populated from natural language texts, they offer significant advantages over traditional export formats, such as plain XML.

In particular, ontologies have become a major tool for developing semantically rich applications: Ontology models are capable of representing a large amount of information using a small number of axioms (individuals and relationships). The ontology models provide users with a high level conceptualization of the information, while at the same time enabling them to focus on specific parts of the model. Web ontologies developed in the OWL-DL language also provide a standardized means for querying, linking, and reasoning about knowledge.

As a majority of the world’s knowledge is encoded in natural language text, automating the population of these ontologies using results obtained from NLP analysis of documents (Cimiano, 2006) has recently become a major challenge for NLP applications (Figure 1.1).

![Figure 1.1.: Populating Ontologies from Text](image)
Chapter 1. Introduction to the OwlExporter

1.2. OwlExporter Overview

The OwlExporter is a reusable component capable of populating OWL ontologies from GATE annotations, which is important for users that need to create more expressive models and would like to benefit from the inferences created by a description logic reasoner, such as Racer (Haarslev and Möller, 2001), Pellet (Sirin et al., 2007), or FaCT++ (Tsarkov and Horrocks, 2006).

The General Architecture for Text Engineering (GATE) is a framework used for the processing of natural language documents.\(^1\) The GATE framework can be extended using plug-ins, known as a Collection of REusable Objects for Language Engineering (CREOLE). GATE’s ontology layer is based on OWLIM (Kiryakov et al., 2005), which only supports the less semantically rich OWL-Lite language. In contrast, the OwlExporter makes use of the Protg-OWL libraries and can populate Owl-Lite, OWL-DL, and OWL-Full ontologies.

The OwlExporter is a Processing Resource (PR) that allows to largely automate ontology population from text using a GATE application. With the use of a few simple mapping grammar, the OwlExporter is able to populate an existing ontology using the annotations created by the application (Witte et al., 2010). In addition to domain-specific concepts, the OwlExporter is also capable of exporting NLP concepts, such as Sentence, NP and VP, and integrating reasoning support for entities that reappear within a given text and are linked together using coreference chains.

![Figure 1.2.: The OwlExporter Processing Resource](image)

1.2.1. OwlExporter Features

The main features of the OwlExporter are:

**Exporting individuals:** Create OWL individuals using entities of a corpus.

**Exporting datatype properties:** Create OWL datatype property relationships using information from a corpus.

**Exporting object properties:** Create OWL object property relationships using the relationships found in a corpus.

**Exporting coreference chains:** Create coreference chains using owl:sameAs for entities that re-appear in different parts of a corpus.

\(^1\)GATE, http://gate.ac.uk
1.3. How to read this documentation

To understand how the OwlExporter is able to export entities of a corpus as individuals and relationships of an ontology, we have included an example GATE pipeline, which is discussed in Chapter 2. Also in Chapter 2, we cover the annotations and features needed to export OWL individuals and relationships using the OwlExporter. In Chapter 3, we discuss additional features that are included with the OwlExporter, such as the ability to export entities that reappear in different parts of a corpus as coreference chains. Finally, in Chapter 4, we discuss the process of compiling and importing OwlExporter, as well as the design decisions of our OwlExporter component.
Chapter 2.

Your First Ontology Population Pipeline

In this chapter, we explain the basic use of the OwlExporter, based on the included example application.

2.1. OwlExporter Installation

The goal of the demo is to show how annotations created by a standard NLP pipeline (here ANNIE) can be exported as instances and relationships of an OWL ontology.

**Prerequisites.** In addition to the OwlExporter distribution, you must have GATE installed. Please refer to [http://gate.ac.uk](http://gate.ac.uk) for instructions on how to download and install GATE. Note that you will need GATE version 7 or better to run the OwlExporter.

![](image.png)

**Figure 2.1.: Enabling the Semantic Software Lab repository in GATE’s CREOLE Plugin Manager**

1. Start GATE, open the Plugin Manager (*File → Manage CREOLE Plugins...*)
2. On the fourth tab (*Configuration*), select a writable directory for the local plugin installations and enable the “Semantic Software Lab” repository (see Figure 2.1).
3. On the third tab (*Available to install*), you will now find the OwlExporter System. Select it and click on “Apply All”.


2.1. OwlExporter Installation

4. On the first tab (Installed Plugins), you will now find OwlExporter installed. Select it (either “Load Now” or “Load Always”) and click on “Apply All”. OwlExporter is now ready.

2.1.1. Loading the Demo Application

To load the system, go to File → Ready Made Applications → Semantic Software Lab (see Figure 2.2).1

Figure 2.2.: Loading the OwlExporter pipelines in GATE Developer

For the demo described below we use two example ontologies:

ANNIE Ontology: For the domain ontology we used the demo.owl2 taxonomic classification that models commonly used ontology concepts in GATE such as Person, Organization and Location.

NLP Ontology: A simple NLP ontology that contains common ontology concepts such as Document, Sentence, NP, and relationships such as contains, containsSentence, HEAD and HEAD_START.

2.1.2. Running the Demo Ontology Population Pipeline

In order to populate the ontologies using the demo pipeline (right-click “OwlExporterDemo” → “Run this Application” The results of the pipeline (shown in Figure 2.4) include the newly created annotations needed by the OwlExporter, namely:

1Please refer to the GATE user’s guide, http://gate.ac.uk/sale/tao for further details on working with pipelines in GATE.
2demo.owl is bundled with GATE and can be accessed from the $GATE_HOME/plugins/Ontology_Tools/resources directory.
Chapter 2. Your First Ontology Population Pipeline

Figure 2.3.: The ANNIE and NLP Ontologies

**OwlExportClassDomain:** The OwlExporter uses the OwlExportClassDomain annotation to identify the domain annotations of a document that need to be exported as instances of their related ontology concept in the domain Ontology.

**OwlExportRelationDomain:** The OwlExporter uses the OwlExportRelationDomain annotation to identify the domain annotations of a document that need to be exported as relationships of their related *individuals* in the domain Ontology.

**OwlExportClassNLP:** The OwlExporter uses the OwlExportClassNLP annotation to identify the NLP annotations of a document that need to be exported as instances of their related ontology concept in the NLP Ontology.

**OwlExportRelationNLP:** The OwlExporter uses the OwlExportRelationNLP annotation to identify the NLP annotations of a document that need to be exported as relationships of their related *individuals* in the NLP Ontology.

**OwlExportRelationDomainNLP:** The OwlExporter uses the OwlExportRelationDomainNLP annotation to identify the *object property* relationships that need to be exported between the Domain and NLP ontology.

The output of the pipeline are the two ontologies *domain_out.owl* and *nlp_out.owl*, as specified using the run-time parameters:

- **exportDomainOntology:** Specifies the location of the exported domain ontology.
- **exportNLPOntology:** Specifies the location of the exported NLP ontology.
2.2. OwlExporter Concepts

We now look at the details on how the OwlExporter example pipeline works, so that you understand how to adapt it to your own applications.

2.2.1. Exporting Domain Individuals

In order to export the entities of a document as individuals of the domain ontology, the annotation types created by a pipeline need to be mapped to the related ontology concepts. The `OwlExporterClassDomain` annotation is used to map annotations created by a pipeline to the relevant domain ontology concepts.
Chapter 2. Your First Ontology Population Pipeline

Figure 2.5.: Populated Ontology

```
AnnotationSet as = (gate.AnnotationSet)bindings.get("ann");
Annotation ann = (gate.Annotation)as.iterator().next();
FeatureMap features = ann.getFeatures();
if(ann.getFeatures().get("majorType").toString().compareToIgnoreCase("person_first") == 0) {
    features.put("className", "Person");
}
if(ann.getFeatures().get("majorType").toString().compareToIgnoreCase("location") == 0) {
    String s = ann.getFeatures().get("minorType").toString();
    features.put("className", Character.toUpperCase(s.charAt(0)) + s.substring(1));
}
String in = doc.getContent().getContent(ann.getStartNode().getOffset(), ann.getEndNode().getOffset()).toString();
features.put("instanceName", in);
features.put("representationId", ann.getId());
features.put("kind", "Class");
outputAS.add(as.firstNode(), as.lastNode(), "OwlExportClassDomain", features);
```

Note: The value of the `className` feature for both the OwlExportClassDomain and the OwlExportClassNLP annotations must match the exact name of the mapped-to concept in the ontology.
Both the OwlExportClassDomain and OwlExportClassNLP annotations must contain the features shown in Table 2.1.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>className</td>
<td>This feature is of type String and must match the name of the Concept in the Ontology that the annotated text in the document is an instances of.</td>
</tr>
<tr>
<td>instanceName</td>
<td>This feature is String type and is the normalized form of the annotated text that needs to be exported.</td>
</tr>
<tr>
<td>corefChain</td>
<td>This feature is of type integer and is the ID of the coreference chain.</td>
</tr>
<tr>
<td>representationId</td>
<td>This feature is of type integer and is the ID of the annotation that needs to be exported.</td>
</tr>
</tbody>
</table>

Table 2.1.: Features needed by the OwlExportClassDomain and the OwlExportClassNLP annotations

### 2.2.2. Exporting Domain Datatype Relationships

In order for domain datatype property relationships to be exported, all that is required is the creation of the property in the domain ontology with a name that matches that of the feature of a given annotation that needs to be exported. Additionally, the domain of the datatype property must match the annotation type that the feature belongs to and the range of the property must match the literal type of the information being exported.

```java
features.put("hasGender",
    ann.getFeatures().get("minorType").toString());
```

For instance, continuing with the Person example Jape grammar created previously, for the hasGender information to be exported as a datatype property relationship in the domain ontology, we created the datatype property hasGender that has Person as the domain and xsd:string as the range. Using the features created by the Jape grammar shown above, the hasGender datatype property relationship is created.

**Note:** No additional annotations are needed in order for the OwlExporter to create datatype property relationships.

**Note:** You must make sure the annotation set specified in OwlExporter’s `inputASName` runtime parameter contains both the OwlExportClassDomain, as well as all the original annotations whose IDs are referenced by the `representationId` feature for datatypes to be exported.

### 2.2.3. Exporting Domain Object Property Relationships

In order for domain object property relationships to be exported, the name of the property in the domain ontology must match the `propertyName` feature of the OwlExportRelationDomain annotation. Additionally, the domain of the datatype property must match the annotation type indicated using the `domainId` feature, and the range of the datatype property must match the annotation type indicated using the `rangeId`.

Notes:
- Ensure both OwlExportClassDomain and all original annotations with IDs referenced by the `representationId` feature are included in the input annotation set.
- Use the `classpath` parameter to specify the path to the OWL file containing the model definitions.
- Customize the `classname` parameter to generate OWL classes based on the annotation types.
- The Class2OWL file(s) are generated in the output directory specified by the `outputDir` parameter.
Chapter 2. Your First Ontology Population Pipeline

Both the **OwlExportRelationDomain** and **OwlExportRelationNLP** annotations must contain the features shown in Table 2.2.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>propertyName</td>
<td>This feature is of type String and must match the name of the related object property in the ontology.</td>
</tr>
<tr>
<td>domainId</td>
<td>This is a feature of type Integer, it is the id of the annotation that is set as the domain of the related object property.</td>
</tr>
<tr>
<td>rangeId</td>
<td>This feature is also of type Integer, it is the id of the annotation that is set as the range of the related object property.</td>
</tr>
<tr>
<td>kind</td>
<td>This feature is of type String and is used by the OwlExporter to identify annotations that need to be exported as relationships. The value of this feature is always JAPE.</td>
</tr>
</tbody>
</table>

**2.2.4. Exporting NLP Individuals**

In order to export the entities of a document as individuals of the NLP ontology, the annotation types created by a pipeline need to be mapped to the related ontology concepts. The **OwlExporterClassNLP** annotation is used to map annotations created by a pipeline to the relevant NLP ontology concepts – see Figure 2.6 for the populated NLP ontology for our example text.
2.2. OwlExporter Concepts

Figure 2.6.: Populated NLP Ontology

```java
String in = doc.getContent().getContent(ann.getStartNode().getOffset(), ann.getEndNode().getOffset()).toString();

if(ann.getType().toString().compareToIgnoreCase("Document") == 0) {
    features.put("instanceName", ann.getFeatures().get("title");
}
else if (ann.getType().toString().compareToIgnoreCase("Sentence") == 0) {
    features.put("content", in);
    features.put("instanceName", in);
    features.put("beginLocation", ann.getStartNode().getOffset());
    features.put("endLocation", ann.getEndNode().getOffset());
}
else if (ann.getType().toString().compareToIgnoreCase("NP") == 0) {
    features.put("instanceName", in);
    features.put("HEAD", ann.getFeatures().get("HEAD");
    features.put("HEAD_END", ann.getFeatures().get("HEAD_END");
    features.put("HEAD_START", ann.getFeatures().get("HEAD_START");
}
outputAS.add(as.firstNode(), as.lastNode(), "OwlExportClassNLP", features);
Chapter 2. Your First Ontology Population Pipeline

2.2.5. Exporting NLP Datatype Relationships

Exporting NLP datatype properties is similar to exporting domain datatype properties. The only difference being the creation of the property in the NLP ontology with a name that matches that of the feature of a given annotation that needs to be exported.

For instance, continuing with the NP example Jape grammar created previously, for the HEAD information to be exported as a datatype property relationship in the NLP ontology, we created the datatype property HEAD that has NP as the domain and xsd:string as the range. And using the features created by the Jape grammar shown above, the HEAD datatype property relationship is created.

Note: You must ensure the annotation set specified in OwlExporter's inputASName run-time parameter contains both the OwlExportClassNLP and all the original annotations whose IDs are referenced by the representationId feature for datatypes to be exported.
2.2.6. Exporting NLP Object Property Relationships

The process of exporting NLP object property relationships is similar to exporting domain object property relationships. The name of the property in the NLP ontology must match the propertyName feature of the OwlExportRelationNLP annotation. Additionally, the domain of the datatype property must match the annotation type indicated using the domainId feature, and the range of the datatype property must match the annotation type indicated using the rangeId.

```java
1 gate.AnnotationSet sentAS = 2 (gate.AnnotationSet)bindings.get("ann");
3 gate.Annotation sentAnn = 4 (gate.Annotation)sentAS.iterator().next();
5 AnnotationSet npAS = 6 inputAS.getContained(sentAS.firstNode().getOffset(),
7 sentAS.lastNode().getOffset()).get("NP");
8 String propertyName="contains";
9 for(Annotation a : npAS) {
10   gate.FeatureMap features = Factory.newFeatureMap();
11   features.put("propertyName",propertyName);
12   features.put("domainId", sentAnn.getId());
13   features.put("rangeId",a.getId());
14   features.put("kind", "JAPE");
15   outputAS.add(npAS.firstNode(), npAS.lastNode(),
16     "OwlExportRelationNLP",features);
17 }
```

2.2.7. Exporting Object Property Relationships across Domain and NLP Ontologies

The OwlExporter is also capable of exporting relationships between the domain and NLP ontologies. Using the demo pipeline, we create an object property relationship appearsIn that links the domain entities to the sentences they appear in. We begin by modifying the domain ontology to import the NLP ontology. This enables us to create the appearsIn relationship in the domain ontology that has Person, Location, and Organization from the domain ontology as the domain and Sentence from the NLP ontology as the range.

```java
1 gate.AnnotationSet sentAS = 2 (gate.AnnotationSet)bindings.get("ann");
3 gate.Annotation sentAnn = 4 (gate.Annotation)sentAS.iterator().next();
5 AnnotationSet domAS = 6 (gate.AnnotationSet)bindings.get("ann");
7 AnnotationSet domAS = 8 (gate.AnnotationSet)sentAS.iterator().next();
```
Chapter 2. Your First Ontology Population Pipeline

```java
    inputAS.getContained(sentAS.firstNode().getOffset(),
    sentAS.lastNode().getOffset()).get("MentionDomain");
    gate.Annotation domAnn =
    (gate.Annotation)domAS.iterator().next();

    String propertyName="appearsIn"
    for(Annotation a : domAS) {
        gate.FeatureMap features = Factory.newFeatureMap();
        features.put("propertyName",propertyName);
        features.put("domainId", a.getFeatures().
        get("representationId");
        features.put("rangeId",sentAnn.getFeatures().
        get("representationId");
        features.put("kind", "JAPE");
        outputAS.add(sentAS.firstNode(), sentAS.lastNode(),
        "OwlExportRelationDomainNLP",features);
    }
```
Chapter 3.

OwlExporter Reference

In this chapter, we cover some advanced use cases of the OwlExporter.

3.1. OwlExporter Run-time Parameters

Table 3.1 documents all run-time parameters of the OwlExporter.

<table>
<thead>
<tr>
<th>Run-time Parameters</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>corefChainList</td>
<td>ArrayList</td>
<td></td>
<td>The set of coreference chains in the text corpus.</td>
</tr>
<tr>
<td>debugFlag</td>
<td>Boolean</td>
<td>●</td>
<td>Run the OwlExporter with or without the debugger. When set to true the</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>OwlExporter prints various messages during different processing stages.</td>
</tr>
<tr>
<td>exportDomainOntology</td>
<td>URL</td>
<td>●</td>
<td>The location and filename of the outputted domain ontology.</td>
</tr>
<tr>
<td>exportNLP</td>
<td>boolean</td>
<td>●</td>
<td>Export an NLP ontology or not.</td>
</tr>
<tr>
<td>exportNLPOntology</td>
<td>URL</td>
<td>●</td>
<td>The location and filename of the outputted NLP ontology.</td>
</tr>
<tr>
<td>importDomainOntology</td>
<td>URL</td>
<td>●</td>
<td>The path of the file that contains the pre-existing domain ontology that the</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>OwlExporter will use to populate and create the exportDomainOntology output</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ontology.</td>
</tr>
<tr>
<td>importNLPOntology</td>
<td>URL</td>
<td>●</td>
<td>The path of the file that contains the pre-existing NLP ontology that the</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>OwlExporter will use to populate and create the exportNLPOntology output</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ontology.</td>
</tr>
<tr>
<td>inputASName</td>
<td>String</td>
<td></td>
<td>The processing annotation set where the OwlExporter will look for the</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>OwlExportClass and OwlExportRelation.</td>
</tr>
<tr>
<td>multiOwlExport</td>
<td>Boolean</td>
<td>●</td>
<td>Output a single Owl file or multiple Owl Files that are all imported by a</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>base output ontology.</td>
</tr>
</tbody>
</table>

Table 3.1.: Summary of OwlExporter Run-time Parameters

3.2. Advanced Features

3.2.1. Document Annotation

In order for the OwlExporter to be aware of the documents that were already processed, a Document concept can be created in either the domain or NLP ontology that contains
Chapter 3. OwlExporter Reference

document instances that store the information of the documents that have been processed. For the exact location of the document, a sourceUrl datatype property is created that stores the url location of the processed document. In Figure 3.1, we show an example of a JAPE grammar that creates a Document annotation that includes the required features shown below:

**title**: Used to store the name of the document  
**sourceUrl**: Used to store the source url of the document

```java
try {
    FeatureMap features = Factory.newFeatureMap();
    features.put("title", doc.getName());
    features.put("sourceUrl", doc.getSourceUrl().getFile());
    outputAS.add((long) 0, doc.getContent().size(), "Document", features);
} catch (InvalidOffsetException ioEX) {
    System.out.println(ioEX);
}
```

Figure 3.1.: An Example of the Doc Info Grammar

*Note:* The OwlExporter uses the title and sourceUrl information to track which documents have already been processed. If a document has already been processed, a warning message will be displayed.

We modified the NLP Ontology by adding a Document concept that the Document annotations in the GATE application will be exported to as individuals of the ontology.

*Note:* The Document instance can be exported in either the Domain or NLP ontology depending on the design of the ontology.

### 3.2.2. Exporting Coreference Chains

The OwlExporter also supports modelling entities that reappear in different parts of a corpus, and that are linked together using coreference chains. For example, a coreferencer such as the one described by [Witte and Bergler, 2003] can identify the nominal and pronominal coreferences between entities and create chains that link them together. When the coreferencer identifies entities of a corpus as being part of the same referent or representative an annotation (for example, Lookup Chain) is created that contains a list of entities that make up a coreference. The OwlExportClassDomain and OwlExportClassNLP annotations accept a corefChain feature that contains the ID of the coreference chain.

```java
AnnotationSet chain = annotations.get("Lookup Chain");
Annotation ann = (gate.Annotation)as.iterator().next();
FeatureMap features = ann.getFeatures();
```
3.2. Advanced Features

By including the corefChain feature the OwlExporter creates:

**The corefSentenceWithId object property relationship:** The corefSentenceWithId relationship associates the referent in a chain with the sentences containing the occurrences of the coreferences.

**The corefStringWithId relationship:** The corefStringWithId relationship ties the referent to the multiple occurrences of its coreferences.

**sameAs relationship:** The OwlExporter establishes the links between the individuals in the ontology using the symmetric, transitive and reflexive owl:sameAs property (Franz Baader et al., 2007). This allows the related individuals to be classified by an OWL reasoner as being equivalent.

```java
if(chain==null||chain.isEmpty())
{
    System.out.println("chain empty");
    Integer chainId = null;
    Iterator chainIt = chain.iterator();
    while(chainIt.hasNext())
    {
        Annotation chainAnn = (Annotation)chainIt.next();
        if(((ArrayList)(chainAnn.getFeatures().
    get("IDs"))).contains(ann.getId()))
            chainId = chainAnn.getId();
    }
features.put("corefChain", chainId);
```
Chapter 4.
OwlExporter Implementation Notes

The OwlExporter is implemented using the Protégé 3.4.2 OWL-API, and the GATE 7 Embedded class library. Note: At present, the installation has been tested under Linux and Windows.

4.1. Compilation

In order to compile the OwlExporter:

- Modify “build.properties” in OWLEXPORTER_HOME to point to the correct GATE_HOME directory on your local machine.

Next, you will need to compile the OwlExporter to include the Protégé libraries needed to run the OwlExporter.

Compiling the OwlExporter:

1. cd to OWLEXPORTER_HOME
2. run ant jar

After performing the steps mentioned above, a Java Archive called OwlExporter.jar is built. This will contain the binaries needed to include the OwlExporter as a CREOLE PR within GATE.

4.2. Importing the OwlExporter

In order to run the OwlExporter, you first need to import the PR into GATE. Plugins are added to GATE using the “Plugin Management Console”. The console can be accessed by either selecting “File - Manage CREOLE plugins” from the menu or the “Manage CREOLE plugins” toolbar item.

When in the “Plugin Management Console” screen, click on the “Add a new CREOLE Repository” button and using the “Open File” dialogue box navigate to the directory containing the OwlExporter.jar file.

After selecting the directory that contains the jar file, the “OwlExporter” plugin will be added to the list of “Known CREOLE Directories”. To begin working with the OwlExporter we will also need to select the “Load now” and/or the “Load Always” options. After completing the
previous steps, click on the “OK” button in order for the OwlExporter to be successfully added to the GATE Developer.

Note: If the installation of the OwlExporter plug-in was unsuccessful, the stack trace displaying the nature of the error can be viewed in the “Messages” tab of the GATE Developer GUI.

4.3. Developer Notes

In this section, we discuss the various requirements taken into consideration when implementing the OwlExporter. Please note that this is not a formal representation of the OwlExporter implementation.

4.3.1. Duplicate Instances

The OwlExporter exports all OwlExportClass annotations from multiple documents and thus there exists the possibility that multiple instances of the same type get exported. We handle this by underscoring the id of the instance (id_x), and we also use owl:sameAs to mark all occurrences of x the same.

4.3.2. Annotation Types versus Annotation Sets

As explained earlier, the OwlExporter only looks for two annotation types, called OwlExportClassDomain/NLP and OwlExportRelationDomain/NLP, to create instances and relationships expressed by object properties, respectively. Handling a temporary annotation makes exporting entities much simpler than, for example, using an ontology-aware JAPE transducer that requires entire input annotation sets. The UI gets simpler because the user doesn’t have to use the GATE UI to add the relevant annotation sets. Authors of components have total flexibility concerning what is exported to the ontology, because only after all their analysis steps are done, they decide which instance (annotation) of their annotation set should become the representation, i.e., the exported individual.
OwlExporter Appendix

A. Main Multi-Phased Jape

```java
MultiPhase: Mention
Phases:
  docinfo
  mention_map_domain_entities
  mention_map_nlp_entities
  mention_map_domain_relation
  mention_map_nlp_relation
  mention_map_domain_nlp_relation
```

B. Doc Info Jape

```java
Phase: docInfo
Input: Token
Options: control = Once
Rule: docInfoRule

({Token}) :ann
```

```java
try{
  FeatureMap features = Factory.newFeatureMap();
  features.put("source",doc.getSourceUrl().getFile());
  features.put("title",doc.getName());
  outputAS.add((long) 0,doc.getContent().size(),
               "Document",features);
}
catch(InvalidOffsetException ioEX) {
  System.out.println(ioEX);
}
```

C. Mention Map Domain Entities

```java
Phase: mention_map_domain_entities
Input: Document Sentence Lookup
Options: control = all debug = true
Rule: mention_map_domain_entities

({Lookup}) :ann
```

```java
try{
  FeatureMap features = Factory.newFeatureMap();
  features.put("source",doc.getSourceUrl().getFile());
  features.put("title",doc.getName());
  outputAS.add((long) 0,doc.getContent().size(),
               "Document",features);
}
catch(InvalidOffsetException ioEX) {
  System.out.println(ioEX);
}
D. Mention Map NLP Entities

try {
    AnnotationSet as = (gate.AnnotationSet)bindings.get("ann");
    Annotation ann = (gate.Annotation)as.iterator().next();
    FeatureMap features = ann.getFeatures();

    if(ann.getFeatures().get("majorType").toString().compareToIgnoreCase("person_first") == 0) {
        features.put("className", "Person");
        features.put("hasGender", ann.getFeatures().get("minorType").toString());
    }

    if(ann.getFeatures().get("majorType").toString().compareToIgnoreCase("location") == 0) {
        String s = ann.getFeatures().get("minorType").toString();
        features.put("className", Character.toUpperCase(s.charAt(0)) + s.substring(1));
    }

    if(ann.getFeatures().get("majorType").toString().compareToIgnoreCase("person_first") == 0 ||
      ann.getFeatures().get("majorType").toString().compareToIgnoreCase("location") == 0) {
        String in = doc.getContent().getContent(ann.getStartNode().getOffset(), ann.getEndNode().getOffset()).toString();
        features.put("instanceName", in);
        features.put("representationId", ann.getId());
        features.put("corefChain", null);
        features.put("kind", "Class");
        outputAS.add(as.firstNode(), as.lastNode(), "OwlExportClassDomain", features);
    }
}
catch(Exception e) {
    e.printStackTrace();
}

D. Mention Map NLP Entities

Phase: mention_map_nlp_entities
Input: Document Sentence NP
Options: control = all
Rule: mention_map_nlp_entities

try {
    AnnotationSet as = (gate.AnnotationSet)bindings.get("ann");
    Annotation ann = (gate.Annotation)as.iterator().next();
    FeatureMap features = Factory.newFeatureMap();
    features.put("className", ann.getType().toString());
    features.put("representationId", ann.getId());
    features.put("corefChain", null);
    features.put("kind", "Class");

    String in = doc.getContent().getContent(ann.getStartNode().getOffset(), ann.getEndNode().getOffset()).toString();
    if(ann.getType().toString().compareToIgnoreCase("Document") == 0) {
        features.put("instanceName", ann.getFeatures().get("title"));
    } else if (ann.getType().toString().compareToIgnoreCase("Sentence") == 0) {
        features.put("content", in);
        features.put("instanceName", in);
        features.put("beginLocation", ann.getStartNode().getOffset());
        features.put("endLocation", ann.getEndNode().getOffset());
    } else if (ann.getType().toString().compareToIgnoreCase("NP") == 0) {
        features.put("instanceName", in);
    }
}
```java
features.put("HEAD", ann.getFeatures().get("HEAD"));
features.put("HEAD_END", ann.getFeatures().get("HEAD_END"));
features.put("HEAD_START", ann.getFeatures().get("HEAD_START"));
}

outputAS.add(as.firstNode(), as.lastNode(), "OwlExportClassNLP", features);
}
}
catch(Exception e) {
e.printStackTrace();
}
}
}
```

### E. Mention Map Domain Relation

**Phase:** mention_map_domain_relation  
**Input:** Token MentionDomain  
**Options:** control = all  
**Rule:** Person_livesIn_Place  

\[
(({\text{MentionDomain className == "Person"}) : \text{dom}}) \rightarrow \text{Person_livesIn_Place} \rightarrow (^{({\text{MentionDomain className == "City"}) : \text{ran}}) \rightarrow ^{({\text{Token.string == "lives"}} {\text{Token.string == "in"}) : \text{rel}}) \rightarrow ^{}}}
\]

```
gate.AnnotationSet domain = (gate.AnnotationSet)bindings.get("dom");
gate.Annotation domainAnn = (gate.Annotation)domain.iterator().next();
gate.AnnotationSet range = (gate.AnnotationSet)bindings.get("ran");
gate.Annotation rangeAnn = (gate.Annotation)range.iterator().next();
gate.AnnotationSet rel = (gate.AnnotationSet)bindings.get("rel");
gate.Annotation relAnn = (gate.Annotation)rel.iterator().next();

String propertyName="livesIn";
```

```java
features.put("propertyName", propertyName);
features.put("domainId", domainAnn.getFeatures().get("representationId"));
features.put("rangeId", rangeAnn.getFeatures().get("representationId"));
features.put("kind", "JAPE");
```

```
outputAS.add(rel.firstNode(), rel.lastNode(), "OwlExportRelationDomain", features);
```

### F. Mention Map NLP Relation

**Phase:** mention_map_nlp_relation  
**Input:** Document Sentence Token Mention  
**Options:** control = all  
**Rule:** Document_containsSentence_Sentence  

\[
(({\text{Document}}) : \text{ann}}) \rightarrow \text{Document_containsSentence_Sentence} \rightarrow ^{}}
\]

```
gate.AnnotationSet docAS = (gate.AnnotationSet)bindings.get("ann");
gate.Annotation docAnn = (gate.Annotation)docAS.iterator().next();
AnnotationSet sentAS = inputAS.getContained(docAS.firstNode().getOffset(),
docAS.lastNode().getOffset()).get("Sentence");
gate.Annotation sentAnn = (gate.Annotation)sentAS.iterator().next();

String propertyName="containsSentence";
```

```java
for(Annotation a : sentAS) {
    gate.FeatureMap features = Factory.newFeatureMap();
    features.put("propertyName", propertyName);
    features.put("domainId", domainAnn.getFeatures().get("representationId"));
    features.put("rangeId", rangeAnn.getFeatures().get("representationId"));
    features.put("kind", "JAPE");
```

```
outputAS.add(sent.firstNode(), sent.lastNode(), "OwlExportRelationDomain", features);
```

---

22
Phase: mention_map_domain_nlp_relation
Input: MentionNLP
Options: control = all

Rule: Domain_appearsIn_Sentence
(MentionNLP.className == "Sentence")

  -->

  gate.AnnotationSet sentAS = (gate.AnnotationSet)bindings.get("ann");
gate.Annotation sentAnn = (gate.Annotation)sentAS.iterator().next();
AnnotationSet domAS = inputAS.getContained(sentAS.firstNode().getOffset(),
  sentAS.lastNode().getOffset()).get("MentionDomain");
gate.Annotation domAnn = (gate.Annotation)domAS.iterator().next();

  String propertyName="appearsIn";
  for(Annotation a : domAS) {
    gate.FeatureMap features = Factory.newFeatureMap();
    features.put("propertyName",propertyName);
    features.put("domainId", a.getFeatures().get("representationId"));
    features.put("rangeId", sentAnn.getFeatures().get("representationId"));
    features.put("kind", "JAPE");
    outputAS.add(sentAS.firstNode(), sentAS.lastNode(), "OwlExportRelationDomainNLP", features);
  }

Rule: Sentence_contains_NP
(Sentence)

  -->

  gate.AnnotationSet sentAS = (gate.AnnotationSet)bindings.get("ann");
gate.Annotation sentAnn = (gate.Annotation)sentAS.iterator().next();
AnnotationSet npAS = inputAS.getContained(sentAS.firstNode().getOffset(),
  sentAS.lastNode().getOffset()).get("NP");

  String propertyName="contains";
  for(Annotation a : npAS) {
    gate.FeatureMap features = Factory.newFeatureMap();
    features.put("propertyName",propertyName);
    features.put("domainId", sentAnn.getId());
    features.put("rangeId", a.getId());
    features.put("kind", "JAPE");
    outputAS.add(npAS.firstNode(), npAS.lastNode(), "OwlExportRelationNLP", features);
  }
Bibliography


